Proceedings of the 18th International Conference on Automation & Computing, Loughborough University, Leicestershire, UK, 8 September 2012

Area and Power optimisation for AES encryption module implementation on FPGA

Tuan Anh Pham, Mohammad S. Hasan and Hongnian Yu
Faculty of Computing, Engineering and Technology
Staffordshire University
Stafford, UK
tapham@live.com, {m.s.hasan, h.yu}@staffs.ac.uk

***Abstract*— Ubiquitous computing has been getting deployed into many applications in daily life. However, one of the difficulties to make it reliable is the lack of security. The constraints of area and power are the challenges for the cryptographic algorithms to be implemented. In this paper, the implementation of Advanced Encryption Standard (AES) encryption algorithm is proposed in terms of resource and power optimisation. The design is based on a 8-bit architecture and implemented on Altera Cyclone II EP2C672C6. The proposed design exhibits 272 LEs and takes 5.88 mW of power which is an improvement in comparison with other published works.**

Keywords: AES-128 encryption, embedded system, FPGA, resource-limited application

I. INTRODUCTION

The popularity of the embedded applications such as smart card, wireless sensor network, RFID technology, etc. has been growing steadily in these recent years. However, the security and privacy issue is one of the most crucial obstructions and concerns to deploy these applications to everyday life usage. Therefore, the requirement of cryptography plays the key role to make these technologies more reliable and effective.

In 2001, a cryptographic algorithm, Advanced Encryption Standard (AES), developed by two Belgian researchers was introduced and approved by NIST [1]. AES provides the strong security, flexible and effective implementation on both software and hardware. Although it is sufficient to perform AES on software, the hardware implementation is preferred because of the high performance and throughput [2], [3].

Generally, the embedded applications do not require very fast speed but have a very limited area and low power consumption constraints [4], [5]. These criteria become the critical significance because they affect directly to the cost and performance of an embedded system. They, therefore, have to be taken into account when implementing the AES algorithm on the hardware.

In this paper, the AES encryption implementation which is operating with the 128-bit cipher key is proposed. The design adopts an 8-bit architecture for entire operations. Further investigations and enhancements are carried out in order to achieve effectively the logic occupancy and power consumption. As the results of the synthesis and compilation the AES-128 encryption design has shown the evident reduction and compactness, where the hardware usage is less than 1% of the chip.

The rest of this paper is formed into six parts. Section II is a brief literature review about AES implementation of ASIC and FPGA. In section III, the proposed AES-128 implementation is presented. Section IV shows the verification and result achievement. A modified version is presented in section V. And the last section, section VI, is the conclusion of the paper.

II. LITERATURE REVIEW

Both Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Array (FPGA) are the interests to realise AES. In ASIC design flows, FPGA is usually used as the prototype for verification before going through the fabrication. However, FPGA has been becoming the modern trend for embedded system demands because of the advantages of the fast time to market, low design cost and reusability [6], [7], [8], [3].

[9], [10], [11], [12], [13] and [14] target for ASIC whilst [15], [8], [7], [16] and [17] target for FPGA. Some of them pay attention on achieving the high throughput of Gbps such as [11], [18], [19]. In these high speed designs, the data are processed in parallel by applying the pipeline technique in each cipher round. Furthermore, the SubBytes transformation is pre-computed and stored in the Ram block which allows the reduction in computation time. Generally, this approach requires many hardware resources which are usually thousands of logic elements.

In contrast, some researchers focus on reducing the area occupancy [9], [13]. The typical method is trying to reuse the components as much as possible and restraining the Ram implementation. From the resource reuse perspective view, only one SBox is implemented and one round is iteratively repeated to complete the encryption. To avoid using the Ram block, the SBox function is represented in the composite field which also benefits to apply the pipeline technique and breakable delay of processing. Other functions, e.g. Key Expansion, are generated on the fly. As the results of those works, the 8-bit data path, though not producing the high operation speed of the circuit, is considered as the most suitable architecture for giving the smallest area and optimised power dissipation.

III. AES IMPLEMENTATION

The AES-128 algorithm designed in this paper is operating in the Electronic Code Book (ECB) mode. Its Register Transfer Level (RTL) architecture is roughly depicted in Figure 1 which comprises of six principle components: Controller, two blocks of Ram 16x8, Rcon, SBox, MixColumns, and Key Expansion. There are two instances of Ram 16x8 blocks that represent the storages for the cipher key and the cipher data. Only one SBox transformation is implemented in order to save the logic elements usage.

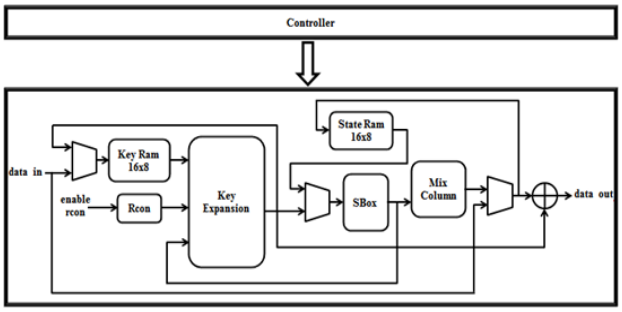

Figure 1. AES encryption architecture

The flow chart of the operation is shown in Figure 2.

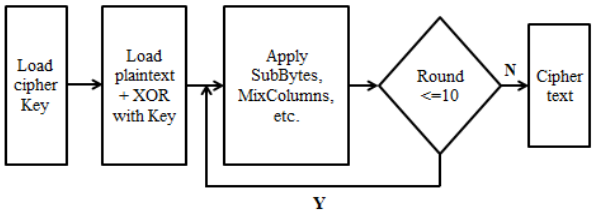

Figure 2. The flow chart of the operation

The cipher key is initially stored in the key ram 16x8. Afterward, the plaintext is XORed with the cipher key to complete the first round and then stored in the State Ram 16x8. In the rest of rounds, the transformations such as SubBytes, MixColumns, etc. are applied to generate the final cipher-text. In each round, 4 bytes of key ram are loaded into 4 registers firstly. And then 4 bytes of state ram are transformed by SBox function and loaded into 4 registers inside MixColumns function. This approach takes 32 clock cycles to encrypt 4 columns of the State array per round.

A. SBox

The direct implementation of SBox on the memory as a lookup table is preferred in high-speed applications. Generally, the full-size AES employs 20 SBoxes to complete the encryption/decryption round where 16 of them are engaged for SubBytes transformation and the remainings are for Key Expansion transformation [16], [17], [18]. However, since the SBox function comes up with much expense of power and hardware resource [20], a memory-based approach is not an appropriate choice in low-area design perspective. Instead, the combinational logic based implementation is adopted, which offers much reduction of area occupancy [20], [19]. Furthermore, in order to save more logic elements, only one SBox is employed in this proposal which is shared between the MixColumns transformation and Key Expansion transformation in turn.

The multiplicative inverse operation of SBox function is calculated in the composite field. This initially requires the isomorphic mapping of the elements in $GF(2^8)$ to the field $GF(((2^2)^2)^2)$ by arithmetically performing the multiplication with the matrix δ. The value after computing the inverse is then converted back to $GF(2^8)$ by multiplying with δ^{-1}. The values of δ and δ^{-1} are shown in 3. TABLE I. displays the irreducible polynomials which are needed to construct the composite fields from the lower order ones.

$$\delta = \begin{pmatrix} 1&0&1&0&0&0&0&0\\ 1&1&0&1&1&1&1&0\\ 1&0&1&0&1&1&0&0\\ 1&0&1&0&1&1&1&0\\ 1&1&0&0&0&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&1&0&1&0&0&1&0\\ 0&1&0&0&0&0&1&1 \end{pmatrix} \quad \delta^{-1} = \begin{pmatrix} 1&1&1&0&0&0&1&0\\ 0&1&0&0&0&1&0&0\\ 0&1&1&0&0&0&0&1\\ 0&1&1&1&0&1&1&0\\ 0&0&1&1&1&1&1&0\\ 1&0&0&1&1&1&1&0\\ 0&0&1&1&0&0&0&0\\ 0&1&1&1&0&1&0&1 \end{pmatrix}$$

Figure 3. Isomorphic mapping and the inverse

TABLE I. FIELD TRANSFORMATION AND CORRESPONDING POLYNOMIALS

Field transformation			Irreducible Polynomial
$GF(2)$	$\rightarrow$	$GF(2^2)$	$P_0(x) = x^2 + x + 1$
$GF(2^2)$	$\rightarrow$	$GF((2^2)^2)$	$P_1(x) = x^2 + x + \phi$
$GF((2^2)^2)$	$\rightarrow$	$GF(((2^2)^2)^2)$	$P_2(x) = x^2 + x + \lambda$

There are 2 different values of ϕ and 8 different values of λ that make $P_1(x)$ and $P_2(x)$ irreducible over $GF(2^2)$ and $GF((2^2)^2)$ respectively [21]. As stated in [22], the choices of ϕ and λ make the impact on complexity of subfield operations and the critical path issue. Therefore, $\phi = \{10\}_2$ and $\lambda = \{1000\}_2$ are opted as one of the optimum results [22].

Let $b = b_1x + b_2$ express the multiplicative inverse of the value $a = a_1x + a_2$ in $GF((2^4)^2)$, where $a_i, b_i \in GF(2^4)$, $i \in \{1, 2\}$. Hence:

$$a.b = 1 \text{ modulo } P2(x) \quad (1)$$

$$\Leftrightarrow (a_1x + a_2)(b_1x + b_2) = 1 \text{ modulo } P_2(x)$$

$$\Leftrightarrow a_1b_1x^2 + (a_1b_2 + a_2b_1)x + a_2b_2 = 1 \text{ modulo } P_2(x)$$

$$\Leftrightarrow a_1b_1(x + \lambda) + (a_1b_2 + a_2b_1)x + a_2b_2 = 1 \text{ modulo } P_2(x)$$

$$\Leftrightarrow (a_1b_1 + a_1b_2 + a_2b_1)x + a_1b_1\lambda + a_2b_2 = 1 \text{ modulo } P_2(x)$$

$$\Leftrightarrow \begin{cases} \\ \end{cases}$$

It is noted that the subtraction operator in the composite field is equivalent to the addition operator. Hence:

$$\Leftrightarrow \begin{cases} b_1 = (a_1a_2 + a_2^2 + a_1^2\lambda)^{-1}a_1 \\ b_2 = (a_1a_2 + a_2^2 + a_1^2\lambda)^{-1}(a_1 + a_2) \end{cases} \quad (2)$$

Deriving from (2), the multiplicative inverse, depicted in Figure 4, can be computed by the combination of a squarer, a multiplier with constant λ, multipliers and an inverter in GF(2^4). These operations can be carried out in the subfields as exposed in TABLE I. The addition in the finite field is simply the bitwise XOR operation. In order to shorten the critical path, a pipeline stage is inserted inside the SBox.

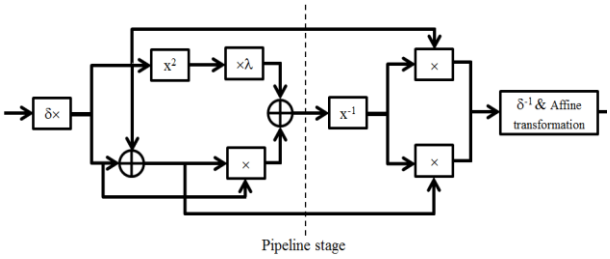

Figure 4. Pipelined SBox architecture

B. MixColumns + SubBytes

The MixColumns and SubBytes transformation are combined into one operation. Instead of applying the SubBytes function to the entire State array, each byte in each column before going to MixColumns is transformed. The 8-bit implementation of MixColumns whose architecture is interpreted Figure 5, requires 4 registers to store 4 bytes of a column and perform the multiplication with the set of coefficients {02, 03, 01, 01}. Although the typical implementation of MixColumns is rotating the coefficients, it is acceptable that the 4 input bytes are iterated but still allow obtaining the same result.

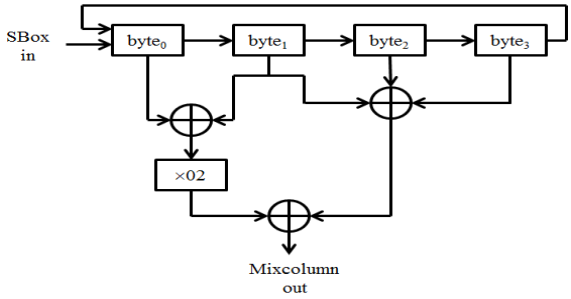

Figure 5. MixColumns architecture

In order to reuse the multiplier, the multiplication of 03 is split into two parts:

$03 = \{11\}_2 = \{10\}_2$ XOR $\{01\}_2 = 02$ XOR 01

Therefore, the first column calculation can be expressed as:

Out = 02×byte0+ 03×byte1+ 01×byte2+ 01×byte3

=02×byte0+02×byte1+01×byte1+01×byte2+ 01×byte3

=02×(byte0+ byte1) + 01×(byte1+ byte2+ byte3)

With this method, the multiplication by 03 is not required any more. Only one multiplier used to compute the multiplication by 02 and 4 XOR operations to perform addition operation is implemented. This leads to an optimised resource usage in comparison with other implementations of MixColumns. An extra logic is also added to get the bypass of MixColumns function when reaching the last round. The multiplication by 02 for the 8-bit input $a_7a_6a_5a_4a_3a_2a_1a_0$ is detailed in equation (3).

$$(a_7a_6a_5a_4a_3a_2a_1a_0<<1) \oplus \{000a_7a_70a_7a_7\} \quad (3)$$

A comparison in term of number of Look Up Table (LUTs) with [23] and another design reported in their paper is resulted in TABLE II.

TABLE II. COMPARISON RESULT OF MIXCOLUMNS IMPLEMENTATION

	Number of logic elements
Proposed MixColumns	36
[23]	43
[24]	48

C. Key Expansion

Key Expansion is an independent routine that generates the round key for encryption/decryption process. Its architecture is described in Figure 6. As mentioned in section IIIA, Key Expansion transformation shares the SBox function with MixColumns transformation. Therefore, four internal registers are needed to store the four bytes of the round key after transformed by SBox..

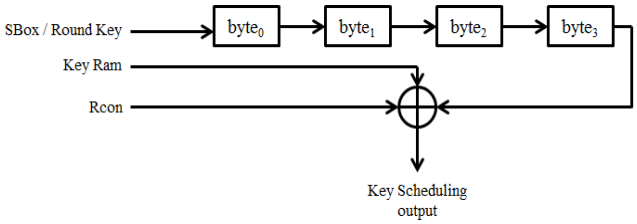

Figure 6. Key Expansion architecture

The round key are generated on the fly, rather than pre-computing and storing in RAM block as introduced in [4]. This method reduces the memory usage and power consumption since the circuit is only activated when needed.

D. Rcon

Rcon is the circuit that generates the round constants for Key Expansion and is constructed in form of $\{(x)^{i-1}, (00), (00), (00)\}$, where *i* starting from 1. It can be implied that only the first byte of each element of key schedule array is affected by the Rcon function. Thus this function is only activated one time per round.

E. Ram 16x8

There are two dual-port Ram 16x8 blocks implemented; one is used to store the cipher key (key ram) during key scheduling process and the other is to hold the State array elements (state ram) for the round transformations. Since the SubBytes, MixColumns and Key Expansion transformation are performed on the fly, there is no requirement for any extra RAM storage.

F. Controller

The operation of the AES-128 core is maintained by a controller which is realised as a finite state machine (FSM) which is shown in Figure 7.

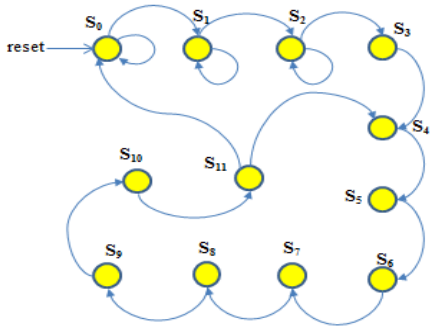

Figure 7. The states of FSM

The entire process of encryption goes through 12 states to produce the cipher text. At each state, the related control signals are triggered to correlate the corresponding transformation functions. The state is encoded using one-hot coding style to improve the efficiency of the power usage and restrain the undesired glitch [25], [9], [26].

Furthermore, the ShiftRows transformation is also completed inside the controller by providing the appropriate read/write address to Ram 16x8 blocks. For the key ram, the address is directly generated by a 4-bit counter. However, for the state ram, it is not simply an up-counter. The analysis on the address generation show that it iterates every four steps as sketched in Figure 8.

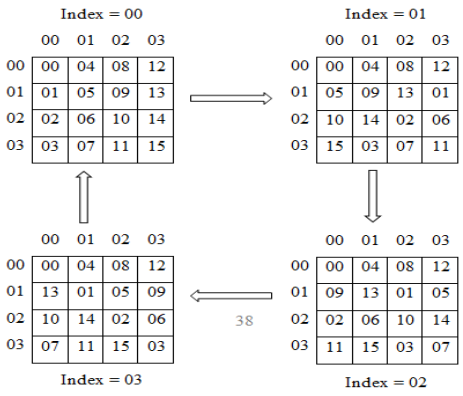

Figure 8. State ram address generation

Each cell address in the State array can be expressed by the equation (4). All variables in this equation are 2-bit length. Hence, the variables row and column are in range of [0…3]. The value of coeff in former calculation is reused for next turn.

if (row=00) coeff = column; else coeff = coeff + index

$$\text{cell\_address}_{\text{row,column}} = \text{row} + \text{coeff} \times 2^2 \quad (4)$$

Each byte of the State array is rewritten to the same address where it was read from after transformation and adding with the round key.

IV. VERIFICATION AND RESULTS

A. Verification method

The AES-128 design is implemented in Verilog Hardware Description Language. The Web Edition version 11.1 of Altera Quartus II is associated with the low-cost target device FPGA Cyclone II EP2C35F672C6 which is mainly used for executing the routines of synthesis, place-and-route and netlist generation.

The netlist generated by Quartus software is then verified by Modelsim Altera Starter Edition. The input vectors of the cipher key and plaintext are adopted from [1]. Power consumption is measured by Altera PowerPlay Power Analyzer tool. The Xilinx ISE Design Suite 13.4 is also employed to fairly make the resource usage comparison with other proposals.

B. Resulst for power consumption

Power consumption of the circuit is taken into consideration when designing the AES-128. FPGA chip power is composed of the dynamic and static parts [27]. Since static power depends much on the transistor technology and operating temperature, it is not feasible to control at system level. Therefore, the dynamic power caused by the switching activity of the signal is considered in order to reduce the power dissipation.

The first technique applied is one-hot coding for the finite state machine. This technique ensures two-bit transition when the state changes. Secondly, an investigation on SBox operation indicates that SBox function is not necessary in 16 clock cycles in each round. This leads to the fact that it can be turned off in 160 clock cycles to save the power. Therefore, the clock-gating technique is applied at the pipeline stage of the SBox to deactivate its functionality. In order to measure power usage, Modelsim Altera Starter Edition software is used to generate the Value Change Dump (vcd) file which is the input for Altera PowerPlay Power Analyzer tool. It should be noted that the ambient temperature is set at the constant of 25ºC. The dynamic power consumption is given in Figure 9 and the comparisons with other designs are given in TABLE III. It can be seen that the proposed implementation achieves significant reduction in total dynamic power dissipation.

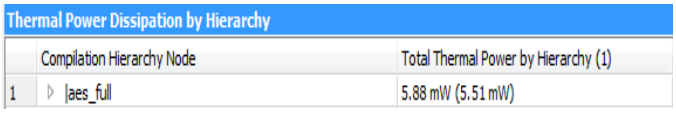

Figure 9. The power consumption of proposed design

TABLE III. COMPARISON IN POWER CONSUMPTION

	AES core dynamic power consumption (mW)
[15]	14.0253
[17]	21.87
[23]	708.68
Proposed design	**5.88**

C. Resulst for area and thoughput

The resource utilisation of the AES-128 core is shown in Figure 10. The total logic elements occupied on Cyclone II EP2C35F672C6 is 272 including the controller which is less than 1% of the chip area. Two simple dual-port Rams used to store the cipher key and State array consume 128 memory bits each. There is a total of 20 I/O pins dedicated for the input and output. As seen from Figure 10, SBox takes the most resource area. The part

"Others" in the pie chart explains the multiplexers and some glue logics to control the data path.

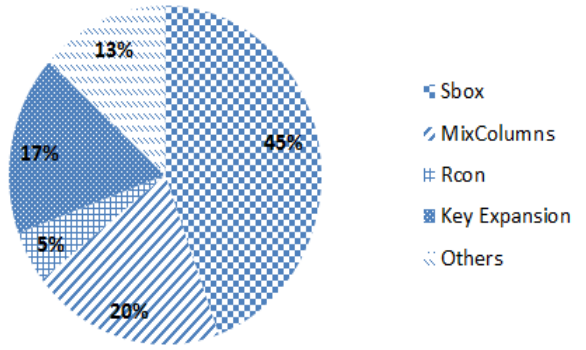

Figure 10. Area occupied by the resource components

The encryption process requires 352 clock cycles to yield the cypher text. The maximum operation clock obtained in this design is 78.39 MHz which results the throughput of 28.5 Mbps. In order to compare with other researches in term of area usage, the proposed design is synthesised in Xilinx ISE tool with similar target devices. Furthermore, instead of using BRAM, the tool is configured to employ the slices only in order to calculate the total equivalent logics. The comparison results are shown in Figure 11.

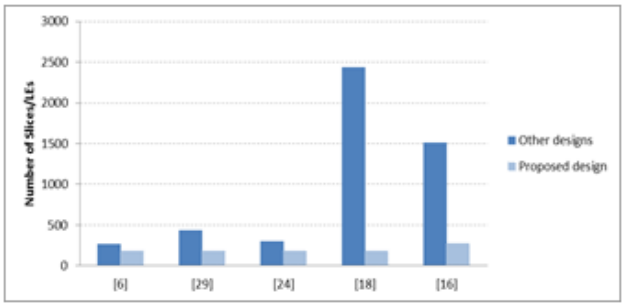

Figure 11. The comparisons of logic occupancy

In comparison with [7], [24], [28], the proposed AES-128 reduces the logic occupancy about 30% to 100%. Compared to the designs of [17] and [15], the resource usage is significantly lower since those designs are implemented on 128-bit architecture and they use memory to store the outputs of SubBytes transformation.

The comparisons in term of efficiency are exposed in Figure 12. The equation to compute efficiency can be derived from equation (5). It should be noted that not all of the designs presented in area comparison are compared here because some of them are lack of information to calculate the efficiency.

Efficiency= Throughput/logic resource = (5)
(128*clock speed) / (clock cycles*no. of logics)

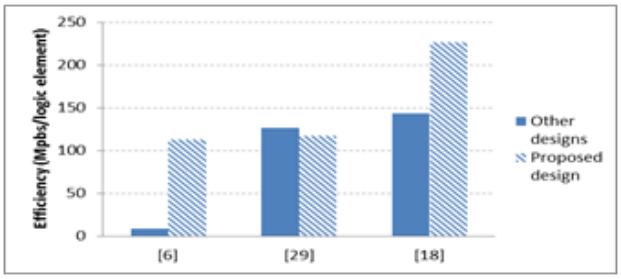

Figure 12. The comparisons of efficiency

V. MODIFIED MIXCOLUMNS

In the MixColumns function of the proposed architecture, 4 registers are used to store the values of each columns of the State array. This approach however requires 4 clock cycles just to reload the new columns after successfully processing the current one and takes 16 clock cycles in one round. In order to improve the throughput, another 4 registers are inserted. The architecture of the modified MixColumns is depicted in Figure 13.

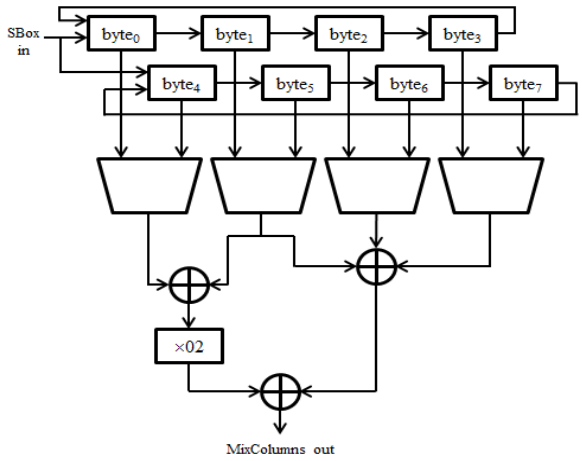

Figure 13. The modified MixColumns

This design makes use of the period when the current column is being processed; the next column will be loaded and then continuously carried out in next turn. Consequently, each round can be completed in 16 clock cycles in comparison with 32 cycles in the original approach. Obviously, the logic element occupancy increases as shown in TABLE IV. It can be observed that the modified version offers higher efficiency due to much clock cycle decrease (31.25%) but little resource usage increase (18.77%). The maximum operating frequencies of two designs are very similar.

TABLE IV. COMPARISON BETWEEN ORIGINAL AND MODIFIED VERSIONS

	Original design	**Modified design**
Mixcolumns	36 LEs	75 LEs
Total LEs	277	329
Clock cycle	352	242
Fmax (MHz)	78.39	72.42
Efficiency	102.9	116
Power (mW)	5.88	6.38

VI. CONCLUSION

In this paper, an optimised area and power implementation of the AES-128 encryption algorithm is presented on FPGA. Regarding the constraints of resource occupancy and low-power requirement, the design is developed on the 8-bit architecture and adopts some techniques to reduce the power consumption, e.g. one-hot coding, clock gating. The results of the simulation and verification have shown a very compact circuit of 277 logic elements and 5.88mW energy dissipation. The proposed architecture is also compared with other published designs in order to demonstrate the effectiveness and improvement of the proposal. In addition, a modified version is presented which offers a higher speed and better efficiency in comparison with the original design but it takes higher logic usage.

ACKNOWLEDGE

This work has been supported by the European Erasmus-Mundus Sustainable eTourism project 2010-2359, the EPSRC UK-Japan Network on Human Adaptive Mechatronics Project (EP/E025250/1) and EU Erasmus Mundus Project-ELINK (EM ECW-ref.149674-EM-1-2008-1-UK-ERAMUNDUS).

REFERENCES

[1] National Institute of Standards and Technology (NIST). (2001, 05 Dec 2011). Announcing the ADVANCED ENCRYPTION STANDARD (AES). FIPS 197. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] S. M. Farhan, S. A. Khan, and H. Jamal, "An 8-bit systolic AES architecture for moderate data rate applications," Microprocess. Microsyst., vol. 33, pp. 221-231, 2009.

[3] A. M. Deshpande, M. S. Deshpande, and D. N. Kayatanavar, "FPGA implementation of AES encryption and decryption," in Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on, 2009, pp. 1-6.

[4] G. Rouvroy, F. X. Standaert, J. J. Quisquater, and J. D. Legat, "Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," in Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on, 2004, pp. 583-587 Vol.2.

[5] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," in Cryptographic Hardware and Embedded Systems - CHES 2003. vol. 2779, C. Walter, Ç. Koç, and C. Paar, Eds., ed: Springer Berlin Heidelberg, 2003, pp. 319-333.

[6] N. C. Iyer, P. V. Anandmohan, D. V. Poornaiah, and V. D. Kulkarni, "High Throughput, low cost, Fully Pipelined Architecture for AES Crypto Chip," in India Conference, 2006 Annual IEEE, 2006, pp. 1-6.

[7] T. Good and M. Benaissa, "Very small FPGA application-specific instruction processor for AES," Circuits and Systems I: Regular Papers, IEEE Transactions on, vol. 53, pp. 1477-1486, 2006.

[8] P. B.Ghewari, M. J. K.Patil, and A. B.Chougule, "Efficient Hardware Design and Implementation of AES Cryptosystem," International Journal of Engineering Science and Techonology, vol. 2, pp. 213-219, 2010.

[9] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES implementation on a grain of sand," Information Security, IEE Proceedings, vol. 152, pp. 13-20, 2005.

[10] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," presented at the Proceedings of the 9th EUROMICRO Conference on Digital System Design, 2006.

[11] H. Li, "Efficient and flexible architecture for AES," Circuits, Devices and Systems, IEE Proceedings -, vol. 153, pp. 533-538, 2006.

[12] S. K. R. S, R.Sakthivel, and P. Praneeth, "VLSI Implementation of AES Crypto Processor for High Throughput," INTERNATIONAL JOURNAL OF ADVANCED ENGINEERING SCIENCES AND TECHNOLOGIES, vol. Vol.6, pp. 22-26, 2011.

[13] T. Good and M. Benaissa, "692-nW Advanced Encryption Standard (AES) on a 0.13-um CMOS," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 18, pp. 1753-1757, 2010.

[14] A. Ricci, M. Grisanti, I. De Munari, and P. Ciampolini, "Design of a 2uW RFID baseband processor featuring an AES cryptography primitive," in Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on, 2008, pp. 376-379.

[15] L. Ai-Wen, Y. Qing-Ming, and S. Min, "Design and implementation of area-optimized AES based on FPGA," in Business Management and Electronic Information (BMEI), 2011 International Conference on, 2011, pp. 743-746.

[16] C. Chiţu and M. Glesner, "An FPGA implementation of the AES-Rijndael in OCB/ECB modes of operation," Microelectronics Journal, vol. 36, pp. 139-146, 2005.

[17] J. Van Dyken and J. G. Delgado-Frias, "FPGA schemes for minimizing the power-throughput trade-off in executing the Advanced Encryption Standard algorithm," Journal of Systems Architecture, vol. 56, pp. 116-123, 2010.

[18] J. M. Granado-Criado, M. A. Vega-Rodríguez, J. M. Sánchez-Pérez, and J. A. Gómez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration," Integration, the VLSI Journal, vol. 43, pp. 72-80, 2010.

[19] M. H. Rais and S. M. Qasim, "Efficient Hardware Realization of Advanced Encryption Standard Algorithm using Virtex-5 FPGA," International Journal of Computer Science and Network Security, vol. Vol. 9, 2009.

[20] N. Ahmad, R. Hasan, and W. M. Jubadi, "Design of AES S-box using combinational logic optimization," in Industrial Electronics & Applications (ISIEA), 2010 IEEE Symposium on, 2010, pp. 696-699.

[21] D. Canright, "A Very Compact S-Box for AES

[22] Cryptographic Hardware and Embedded Systems – CHES 2005." vol. 3659, J. Rao and B. Sunar, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 441-455.

[23] Z. Xinmiao and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," Circuits and Systems II: Express Briefs, IEEE Transactions on, vol. 53, pp. 1153-1157, 2006.

[24] S. Ghaznavi, C. Gebotys, and R. Elbaz, "Efficient Technique for the FPGA Implementation of the AES MixColumns Transformation," presented at the Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs, 2009.

[25] L. Hua and Z. Friggstad, "An efficient architecture for the AES mix columns operation," in Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on, 2005, pp. 4637-4640 Vol. 5.

[26] G. Sutter and E. Boemo, "Experiments in low power FPGA design," Latin American applied research, vol. 37, pp. 99-104, 2007.

[27] C. E. Cummings, "The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates," in International Cadence Usergroup, San Jose, California, 2002.

[28] P. P. Czapski and A. Sluzek, "A Survey on System-Level Techniques for Power Reduction in Field Programmable Gate Array (FPGA)-Based Devices," in Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on, 2008, pp. 319-328.

[29] Z. Bin, P. Yingning, K. Gaj, and Z. Zhonghai, "Implementation and comparative analysis of AES as a stream cipher," in Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, 2009, pp. 396-400.